VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2016 Rostislav Vondrák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Rostislav Vondrák**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. RNDr. Petr Šaloun, Ph.D.**

Konzultant bakalářské práce: Ing. Lukáš Zajac

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

Tieto Czech s.r.o.
28. října 3346/91
702 00 Ostrava - Moravská Ostrava
IČO 64608051 DIČ CZ64608051

V Ostravě 29. dubna 2016 .....

Rád bych poděkoval firmě Tieto Czech s.r.o. za možnost absolvování odborné praxe. Zvláště pak bych rád poděkoval Ing. Lukáši Zajacovi a našemu týmu, kteří mi vždy ochotně poradili.

Dále bych rád poděkoval mému vedoucímu bakalářské práce doc. RNDr. Petru Šalounovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této bakalářské práce.

Abstrakt

Tato bakalářská práce popisuje mnou řešené úkoly, které jsem dostával během odborné praxe ve firmě Tieto Czech s.r.o. Ve firmě jsem pracoval na projektu informačního systému pro energetické společnosti. První část práce popisuje detaily informačního systému, které jsem získal během několika školení ve firmě. V druhé části popisuji řešené úkoly, které byly hlavní náplní mé práce. Všechny mnou řešené úkoly byly opravy chyb, proto se v práci zabývám hlavně analýzou kódu. Tyto chyby jsem řešil v několika různorodých programovacích jazycích, čímž jsem využil vědomosti z mnoha školních předmětů.

Klíčová slova: Tieto Czech s.r.o, informační systém, bakalářská práce, C#, .NET Framework, Oracle SQL, PL/SQL, JavaScript, Visual Basic 6, XML

Abstract

This bachelor thesis describes tasks which I worked on during professional practice in Tieto Czech company. I worked on information system for energy companies. First part of this thesis is deditated to details of information system which I learned on several trainings in the company. Second part describes development tasks which were main part of my work. All solved tasks were classified as error corrections therefore I had to do a lot of code analysis. I solved these errors in various programming languages thereby I used knowledge from many school subjects.

Key Words: Tieto Czech s.r.o, information system, bachelor thesis, C#, .NET Framework, Oracle SQL, PL/SQL, JavaScript, Visual Basic 6, XML

Obsah

Seznam použitých zkratek a symbolů

CAB	–	Customer and Billing
COM	–	Component object model
DML	–	Data manipulation language
GUI	–	Grafické uživatelské rozhraní
IDE	–	Integrated development enviroment (vývojové prostředí)
IS	–	Informační systém
OTI	–	One time invoicing
PL/SQL	–	Procedural Language/Structured Query Language
PTI	–	Pass through invoicing
SI	–	Settlement and Invoicing
SQL	–	Structured Query Language
TFS	–	Team foundation server
VB6	–	Visual Basic 6
VS	–	Visual Studio 2013

Seznam obrázků

1 Úvod

Úvodem této bakalářské práce bych Vás rád seznámil s průběhem mé odborné praxe ve firmě Tieto Czech s.r.o. se sídlem v Ostravě. Tuto alternativní volbu vypracování bakalářské práce jsem si vybral hlavně z důvodu otestování mých programátorských znalostí, které jsem se naučil ve škole v předmětech jako *Programovací jazyky II* a *Architektura technologie .NET*. Dalšími důvody bylo získání praxe v oboru a vyzkoušení si práce v týmu.

Během praxe jsem se musel seznámit s rozsáhlou business logikou, než jsem mohl začít pracovat na dílčích úkolech. Proto se v této práci budu zabývat jak teorií, tak i praxí.

Práce je rozdělena do čtyř kapitol. Třetí kapitola stručně popisuje teoretické znalosti, kterých jsem nabyl v průběhu několika školení ve firmě. Ve čtvrté kapitole popisuji úkoly, které jsem řešil v průběhu praxe. Závěrem práce je zhodnocení mnou získaných zkušeností.

2 Firma Tieto Czech s.r.o

V této kapitole popisuji zaměření firmy, informace o firmě, pracovní zařazení a prostředí, ve kterém jsem po dobu praxe pracoval.

2.1 O Firmě

Firma Tieto Czech s.r.o., viz [1], je předním zaměstnavatelem IT odborníků v Moravskoslezském kraji. Historie firmy sahá do roku 1968, kdy firma vznikla pod názvem Tietotehdas Oy. V roce 1999 se firma Tieto spojila se švédskou společností Enator, přičemž došlo ke změně jména na TietoEnator. V roce 2010 se firma přejmenovala zpět na Tieto. V současnosti Tieto působí v téměř 20 zemích světa a jejími hlavními trhy jsou severní země Evropy.

2.2 Pracovní zařazení

Ve firmě jsem zaujímal pozici C# programátora, nicméně jsem se dostal do styku i s několika dalšími programovacími jazyky. Ze začátku jsem strávil většinu času na školeních se zkušenými mentory, kteří mě zaškolili do business logiky informačního systému CAB. Velkou část mé práce tvoří dílčí úkoly, kdy jde většinou o opravy chyb nahlášené zákazníkem. Abych byl schopen opravovat chyby, musel jsem se často pokusit reprodukovat danou chybu. Z tohoto důvodu jsem musel být schopen také vytvářet testovací data, na kterých se dala simulovat situace u zákazníka.

3 Použité technologie

3.1 .NET Framework

Nejdůležitější technologií, kterou jsem během praxe využíval je *.NET Framework*. *.NET Framework* je technologie od společnosti *Microsoft*, která poskytuje komplexní sadu nástrojů, služeb a prostředí potřebných k vývoji aplikací. Hlavními částmi frameworku je *Common language runtime*, což je prostředí pro běh aplikací, programovací jazyk C# a knihovny, které byly vyvinuty pro *.NET Framework*, viz [2].
Pro vývoj v jazyku C# jsem používal nástroj *Microsoft Visual Studio 2013*, který mi poskytla firma. Ve firmě jsme rovněž využívali *Team Foundation Server 2013*, který nám pomáhal vyvíjet a verzovat sdílený kód a byl součástí *Visual Studia 2013*.

3.2 PL/SQL

Většina výpočtů a validací dat jsou v IS CAB řešeny v databázi. Proto byl PL/SQL druhý nejpoužívanější jazyk při mé práci na IS CAB. PL/SQL je procedurální nádstavba nad SQL, kterou vyvinula firma *Oracle*, viz [3]. Pro práci s PL/SQL a databází samotnou jsem používal IDE *Oracle SQL developer*.

3.3 XML

XML je značkovací jazyk, který popisuje data a používá se pro jejich sdílení mezi různými systémy, viz [4].

3.4 JavaScript

JavaScript je skriptovací jazyk používaný hlavně pro vývoj webových aplikací. V rámci IS CAB se používá hlavně pro vývoj nových grafických rozhraní. Kromě samotného *JavaScriptu* jsem se setkal i s příbuznými technologiemi jako např. *Ajax*, *JSON*, *jQuery*.

3.5 Visual Basic 6

Visual Basic 6 je událostmi řízený programovací jazyk a IDE od společnosti *Microsoft* pro jeho programovací model COM. Je odvozen od programovacího jazyka *BASIC*, viz [5]. V CABu jsou v něm psány uživatelská rozhraní. Již dlouho se v něm nevyvíjí, ale občas je potřeba v těchto komponentách opravit chyby. K vývoji jsem používal *Microsoft Visual Studio 6*.

3.6 CAB Framework

CAB Framework je firemní framework, který kombinuje technologie *ASP.NET*, *JavaScript*, *XML*, *AngularJS* a několika dalších. Důvodem pro jeho vznik bylo, aby architektura, grafické rozhraní i kód měly jednotnou podobu a využívaly jednotných komponent.

4 Informační systém CAB

CAB je informační systém sloužící k evidenci veškerých entit souvisejících s dodávkou energie. Tieto dodává tento IS většině elektrárenských společností z Norska, Finska a Švédska již po několik desítek let. Pro každou společnost má CAB svoje prostředí a u některých společností i v několika verzích. To slouží především k co nejpřesnější simulaci zákaznického požadavku, protože každé prostředí má různá privilegia přístupu k funkcím.
CAB je tlustý klient, který vznikl v roce 1994 a z tohoto důvodu se skládá z mnoha různých jazyků od VB6, přes C a C++ až po současný C#, včetně dalších technologií spojených s *.NET Frameworkem*. Vedle toho jsou použity i webové technologie jako např. *ASP.NET*, *JavaScript*, *AngularJS*. Data pro každé prostředí jsou uložena v *Oracle* databázi ve třech různých databázích. Jen pro zajímavost - každé prostředí databáze momentálně obsahuje 1716 tabulek.
O CAB se stará velké množství odborníků z České republiky, Indie a severských zemí Evropy. V Ostravě na něm pracuje přibližně 100 lidí, převážně programátorů a testerů. Tito lidé jsou rozděleni do sedmi týmů. Náš tým se jmenuje SI a skládá se z 12 lidí, z Indie a České republiky. Naším úkolem je zpracování veškerého fakturování a počítání spotřeby. Tedy hlavně Invoicing, OTI, PTI a Energy pool (vysvětleno níže). Ukázka mé práce v IS CAB je na obr. 1.

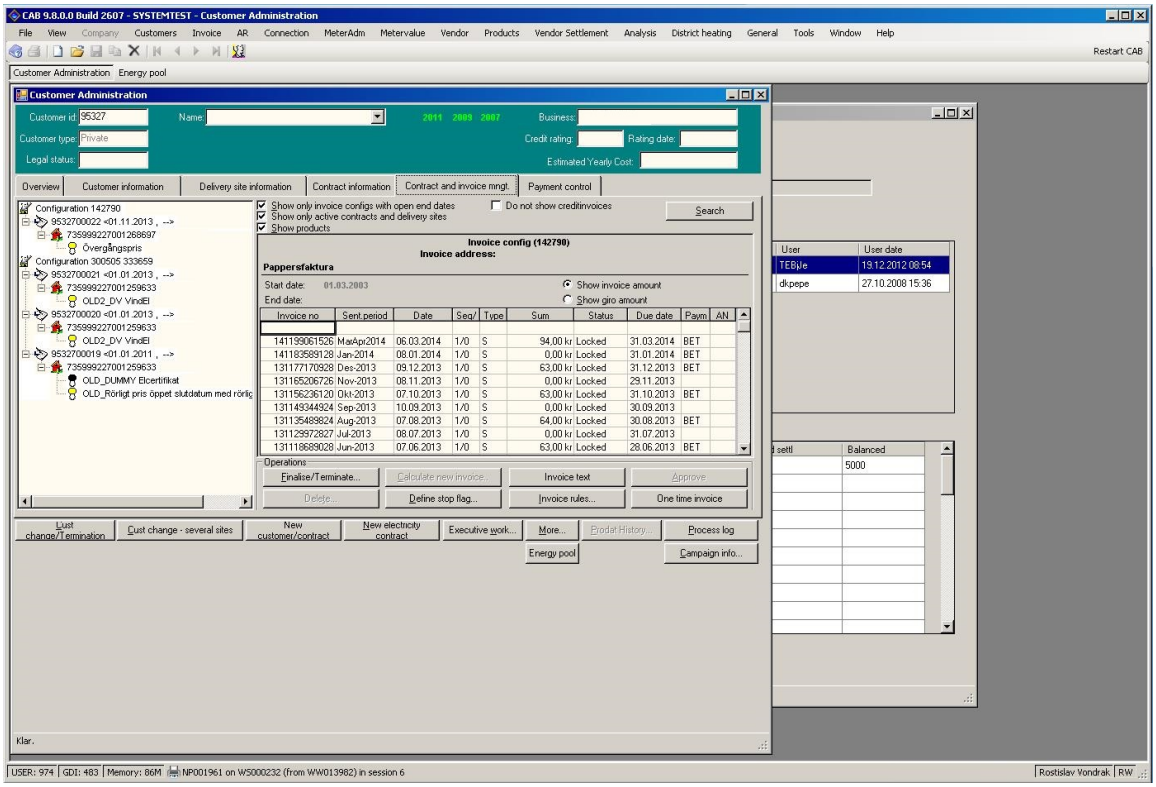
4.1 Customer (Zákazník)

Zákazník představuje entitu odběratele energie. Je to pravděpodobně nejdůležitější část CABu z bussiness pohledu. Při vytváření testovacích dat je zákazník vždy startovacím bodem. Má velmi blízkou souvislost s kontraktem. Zákazník tvoří jakýsi "protějšek"kontraktu pro dodávky energií a ostatních služeb.
Zákazníci se dělí na dva typy: Private (jednotlivce) a Enterprise (firmy). Toto rozdělení zásadně ovlivňuje nabídky produktů, tímpádem i ceny. Mimo toto rozdělení, každý zákazník po vytvoření získává status, který ovlivňuje jeho funkci:

- Prospekt - Zákazník je registrován v CABu, bez jediného kontraktu nebo jeho první kontrakt začíná v budoucnosti
- Potencionální - Zákazník ukončil aktivní kontrakt a nemá žádný další aktivní kontrakt.
- Aktivní - Zákazník má aktivní kontrakt

4.2 Delivery site (Odběrné místo)

Odběrné místo představuje fyzickou budovu, na které je vybudovaná přípojka pro danou službu. Obsahuje informace o přípojce, pojistkách, adresu a je rovněž rozděleno do několika kategorií. Nastavené parametry ovlivňují možnosti dodávání produktů a služeb.
Vzhledem k tomu, že severské země částečně zasahují také do severního polárního kruhu, je



Obrázek 1: Detail zákazníka a jeho kontraktu v IS CAB

potřeba rozlišovat odběrná místa podle adresy. Zákazníci v oblasti severně od polárního kruhu mají rozdílné ceny energií, protože u nich dochází k jevům zvaným polární den a polární noc.

4.3 Meter (Hodiny)

Hodiny jsou nedílnou součástí odběrného místa, které slouží k měření spotřeby služeb (např. energie). Hlavní rozdělení hodin je podle veličiny, kterou hodiny měří. U spotřeby energie jde většinou o spotřebovanou energii v kWh a špičkovou spotřebu. Dalším rozdělením je způsob čtení hodnot z hodin. Mimo klasické opisování hodnot z měřáku, se rovněž používá čtení přes internet nebo SMS.

4.4 Contract (Kontrakt)

Kontrakt je dohoda mezi dodavatelem a zákazníkem. Každý kontrakt má unikátní ID v CABu. Popisuje dodávaný produkt, odběrné místo, jak dlouho bude kontrakt trvat atd. Pro vytvoření kontraktu existují různé předlohy, které definují produkty, platební podmínky, informace o měně a mnoho dalších informací. Každý kontrakt je navíc rozdělen do několika kategorií. Jedná se většinou o rozdělení, co bude daná část kontraktu obsahovat a za co bude zákazník platit např. energie, rozvodné sítě, plyn, vytápění, internet.

4.5 Product (Produkt)

Produkt nebo také služba představuje entitu, podle které se vyúčtovávají faktury zákazníkovi. Je připojen ke konkrétnímu dodavateli. Každý produkt se skládá z jedné nebo více částí produktu. Každá část produktu reprezentuje alespoň jeden řádek ve faktuře. Části se obvykle skládají z Quantity funkce a Price funkce, případně se část produktu skládá jen z Amount funkce. Všechny tři funkce se mohou skládat z jednodušších algoritmů, které pouze načítají cenu a spotřebu z databáze, až po velmi komplexní konstrukce zahrnující množství předdefinovaných základních funkcí ve složitých algoritmech.

4.6 Invoicing (Fakturování)

Invoicing je kolekce aplikací umožňující uživateli vytvářet, příjmat a potvrzovat faktury a zároveň kontrolovat, zda se dané úkony provedly. Každá faktura se skládá z několika položek, které jsou spočítány z objednávek. Každá položka obsahuje jednu část produktu s atributy (resp. měřenými hodnotami). Na faktuře není omezen počet položek a jejich číslování není pevně dáno. Položky na faktuře jsou rozděleny do kategorií, které určují za jaký typ služby je daná položka. Existují i kategorie na zaokrouhlování faktury apod.

4.6.1 Mass invoicing (Hromadné fakturování)

Hromadné fakturování zajišťuje funkcionalita zvaná Automatic Running. Umožňuje hromadně vytvářet, počítat, potvrzovat a tisknout faktury bez nutnosti obsluhy. Uživatel nastavuje podmínky, aby rozlišil, které faktury bude aplikace zpracovávat (např. podle měsíce).

4.6.2 One time invoicing (OTI)

OTI je typ faktury, který není zpracován periodicky, ale pouze jednou. OTI má podobné parametry jako klasická faktura s tím rozdílem, že používá jiné produkty. Obsahuje položky, které definují, za co zákazník bude platit, takže připomínají části produktu. Používá se například k zpracování poplatků a pokut. V konečném důsledku je OTI zpracována souběžně s klasickou fakturou.

4.6.3 Pass through invoicing (OTI)

Principiálně se jedná o obyčejnou fakturu, která ale není vypočítaná u nás. PTI je nový druh faktur, který vznikl z důvodu komunikace CABu s ostatními systémy. Elektrárenské společnosti se dělí na ty, co zpracovávají faktury a ty co vlastní účastnickou síť. Faktury mohou vystavovat pouze fakturovací společnosti. Proto, aby se daná faktura dostala z jednoho systému do druhého, je potřeba ji importovat. Je vypočítána někde jinde a my ji pouze importujeme ve formátu *XML* a následné zaúčtování je vyřešeno vytvořením OTI faktury, která obsahuje data z PTI XML souboru.

4.7 Energy pool

Informace kolik energie (definována pomocí akcií) může zákazník koupit od dodavatele (dodavatel sekundární energie v případě větrné energie). Akcie je právo využívat energii podle aktuálního faktoru (kWh/podíl) po dobu jednoho roku. Zákazník může prodat toto právo používat energii, podle počtu akcií, dalšímu zákazníkovi (všechny akcie nebo jen část). Dodavatel má pouze určitý počet akcií, se kterými může obchodovat (Total energy pool).

5 Zadané úkoly a jejich řešení

V této kapitole budu popisovat mé postupy při řešení jednotlivých úkolů. Všechny úkoly bylo potřeba simulovat na našem prostředí a následně i integračně otestovat. U některých úkolů jsem rovněž musel komunikovat se zákazníkem, protože z nahlášené chyby nebylo někdy poznat, jaké řešení vlastně zákazník požaduje.

5.1 Oprava chyby v přiřazování priorit jednotlivým energy pools

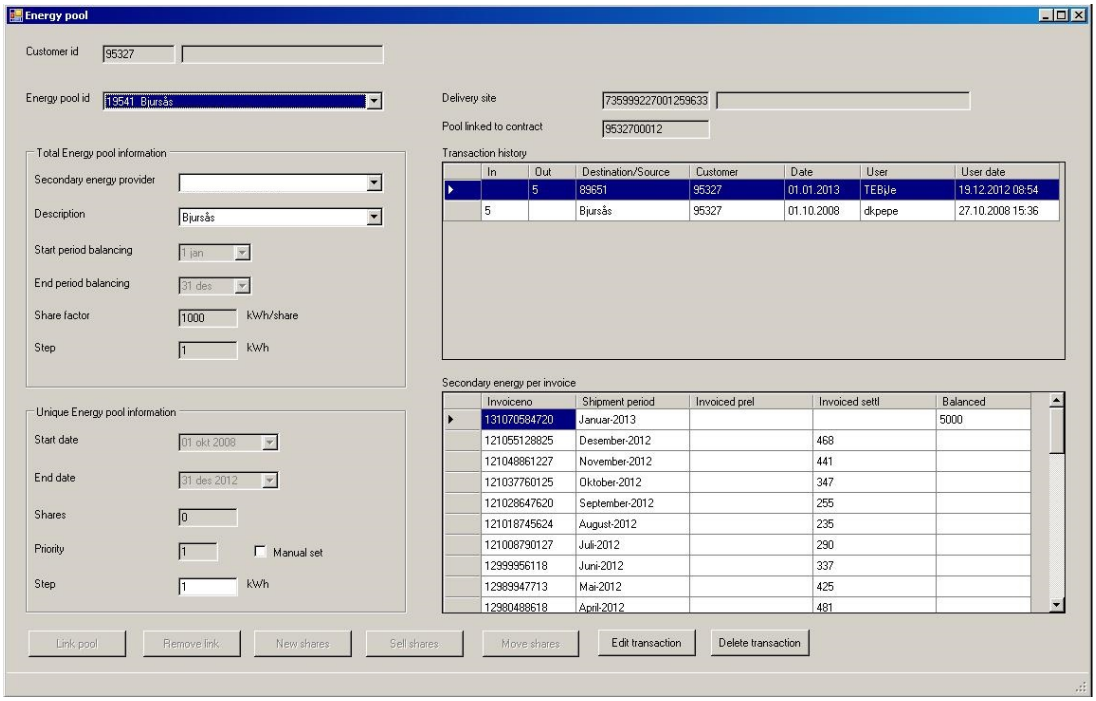
Mým prvním větším úkolem bylo opravit chybu v přiřazování priorit energy pools. Každý energy pool v rámci jednoho odběrného místa musí mít unikátní prioritu v intervalu od 1 do počtu pools (když má odběrné místo 3 pools, pak je interval od 1 do 3). Systém by měl samostatně vypočítat prioritu podle počátečního data. Pokud začínají dva energy pools ve stejný den, pak se přiřadí první volná priorita. Uživatel má možnost nastavit prioritu manuálně, přičemž systém přepočítá priority pro ostatní energy pools.

5.1.1 Popis problému

Při nastavení manuální priority pro energy pool se priorita nastavila a priority pro ostatní se nastavily podle pravidel. Problém nastal, když uživatel zrušil volbu manuálního nastavení. Pro daný energy pool se nastavily nesprávné hodnoty jako -1, 0 nebo null. Pro ostatní se nastavily nesprávné hodnoty, které nesplňovaly pravidla.

5.1.2 Popis řešení

Vzhledem k tomu, že CAB je velmi obsáhlý program, tak hledání daného projektu většinou není možné procházením adresářů v TFS. S hledáním projektu v TFS jsem si pomohl přes *Developer Command Prompt for VS2013*. Pomocí zadání příkazu *tf dir /recursive /$[název]* se dá nalézt jakýkoli soubor v TFS.
Po nalezení projektu jsem analyzoval daný kód a našel jsem hledanou funkci ReorderPriority. Funkce zajišťuje přepočítávání priorit při přidávání a odebírání pools a také při manuálním nastavováním priorit. Je rozdělena do dvou hlavních větví, které rozlišují, zda se jedná o změnu priority pouze pro určitý počet pools nebo se musí přepočítat hodnoty pro všechny pools. Vstupem funkce je identifikátor daného pool a hodnota jeho předchozí priority. V případě přepočtu pouze pro některé pools se rozlišuje, zda se musí přepočítat všechny priority nebo pouze ty vyšší (resp. nižší).
Nově vytvořený pool má prioritu nastavenou na -1, takto funkce pozná, že musí přepočítat všechny priority. Při přechodu z manuálního nastavení priority na automatickou se nebere v potaz původní manuální priorita, ale rovněž se nastavuje priorita na -1.
Chybný algoritmus byl v části, kde se přepočítavají priority pouze pro některé pools, konkrétně v případě, kdy se přepočítavaly pools s větší prioritou.



Obrázek 2: Okno pro správu energy pools

Tuto část kódu jsem přepsal, aby přiřazovala priority postupně od posledního pool. Při tomto řazení se provedla kontrola počátečních dat jednotlivých pools a následně došlo k přiřazení správné priority. Rovněž zde probíhala kontrola na nastavení manuální priority, aby se předešlo změně priority u nesprávných dat. Na konci funkce je kontrola na správně přiřazené priority a případně přiřadí první volnou prioritu. Po dokončení řazení funkce provede commit nad databází a změny jsou uloženy.
Testování jsem prvně prováděl pouze pomocí transakcí nad databází a následně i prostřednictvím GUI, viz obr. 2.

5.2 Oprava chyby v datech Invoice Job

Mým dalším úkolem bylo opravit chybu v přiřazování uživatelského jména z Invoice batch k Invoice job. Pro porozumnění problému popisuji obě entity níže.

5.2.1 Invoice batch

Invoice batch (dávka) je objekt reprezentující hromadné fakturování v CABu. Skládá se z jednoho nebo více dílčích úkolů, které provádějí operace s fakturami. Mezi tyto úkoly patří například vytvoření, vypočtení a tisk. Ke každé dávce je možné přidat jeden nebo více časů plánovaného spuštění.

5.2.2 Invoice job

Jakákoliv operace s fakturami v CABu má za následek vytvoření záznamu v tabulce *invoicejob*, která je v pravidelných intervalech kontrolována Invoice serverem. V tomto záznamu jsou definovaná kritéria a úkoly, které se mají provést s danými fakturami. Tento úkol může být spouštěn vícekrát, aniž by bylo nutné znova vytvářet jeho kopii.

5.2.3 Popis problému

Obě jmenované tabulky obsahují uživatelské jméno toho, kdo ji vytvořil. U Invoice job je to vlastnost *owner*, která uchovává jméno toho, kdo jej vytvořil. Invoice batch má vlastnost *username*, která uchovává uživatele, který provedl poslední změnu. Při prvním spuštění dávky je vytvořen *invoicejob*, který má vlastnost *owner* naplněnou podle *username* z dávky, což je první chyba. Když jsem totiž simuloval situaci, kdy jeden uživatel vytvořil dávku a druhý naplánoval její spuštění, došlo k tomu, že vytvořený invoice job měl nastavený parametr owner na hodnotu prvního uživatele. Další chybou bylo, že při každém naplánování dalšího spuštění dávky by se měl parametr username změnit na jméno toho, kdo jej naplánoval.

5.2.4 Řešení problému

Jelikož jsem nevěděl, ve které části kódu může být chyba, začal jsem s postupnou analýzou kódu od uživatelského rozhraní. GUI je vytvořené v *ASP.NET* s využitím *CAB Frameworku*. V controlleru daného view šlo dohledat, že se při přidání plánovaného spuštění k invoice batch volá ještě databázová procedura SETINVBATCH_TOBERUN. Tato procedura ukládá všechny časy plánovaného spuštění do databázové tabulky INVBATCH_TIMES a nejbližší čas plánovaného spuštění zkopíruje do záznamu v INVBATCH. V tabulce INVBATCH_TIMES je uloženo i správné uživatelské jméno, které se ale nepřenáší do INVBATCH.
Na řadu poté přichází Invoice Server, který zkontroluje, zda neměla být spuštěna nějaká dávka. Pokud ano, pokusí se najít její *invoicejob* a případně vytvoří nový, do kterého zkopíruje username z INVBATCH. Velikým usnadněním je, že Invoice server lze monitorovat pomocí aplikace CMViewer, která loguje co se na Invoice serveru děje. V tomto výpisu je patrné, že byl vytvořen a spuštěn invoice job se špatným uživatelským jménem.
Invoice server je aplikace psaná v C++ s využitím knihoven třetích stran (RogueWave). Jedná se o docela rozsáhlou aplikaci, protože zpracovává veškeré fakturační požadavky z CABu. Po dokončení této analýzy jsem došel k možným místům, kde by se daly opravit chyby.

- V *ASP.NET* controlleru lze aktualizovat tabulku INVBATCH pomocí vykonání jednoduchého DML příkazu nad databází.
- Stejnou možnost aktualizace tabulky INVBATCH lze docílit pomocí úpravy v databázové proceduře SETINVBATCH_TOBERUN.

- Poslední možností bylo upravovat až vytvářený invoice job přímo v Invoice serveru. Tato možnost by, ale neřešila situaci, kdy chceme aktualizovat invoice batch i při změně naplánovaného času.

Nakonec jsem se rozhodl řešit tento problém pomocí úpravy v databázové proceduře, protože se tím ušetří přenášení dat z GUI do databáze (a zpět). K úpravě procedury jsem využil nástroj *Oracle SQL Developer*.
Procedura má jeden vstupní parametr a tím je identifikátor záznamu v INVBATCH. Bylo ji nutné napsat, tak aby při přidání/úpravě/smazání naplánovaného času bylo vždy použito to správné uživatelské jméno. V první části procedury jsem získal naplánovaný čas a uživatelské jméno z tabulky INVBATCH_TIMES a následně jsem upravil záznam v INVBATCH. V druhé části jsem pouze odchytával vyjímku, která nastala při mazání plánovaného času dávky. Tuto vyjímku způsobovala klauzule SELECT INTO v situaci, kdy nebyly nalezeny žádné data. Uživatelské jméno pro INVBATCH jsem tedy získal ze smazaného záznamu v INVBATCH_TIMES.
Po dokončení úprav databázové procedury jsem začal s testováním.
K testování jsem použil funkci Remote debug, kterou nabízí SQL Developer. Pro využití této možnosti bylo potřeba jen znát adresu svého PC. Pomocí funkcí *connect_tcp* a *disconnect* z balíčku *dbms_debug_jdwp* šlo jednoduše specifikovat blok kódu, který se má ladit. Bohužel tato funkcionalita nebyla moc stabilní a velmi často se přerušila bez udání nějaké chybové hlášky.
Proto jsem pro krokování nakonec využil jednoduchého logování do již existující logovací tabulky. Všechny funkce fungovaly dobře, až na jednu specifickou situaci. Když jsem naplánoval dva časy spuštění dávky od dvou různých uživatelů (*Honza* a *Karel*), tak poté co invoice server spustil dávku poprvé, zůstalo v tabulce INVBATCH jméno uživatele z prvního spuštění (*Honza*), i když se datum naplánovaného spuštění změnilo. Takže při druhém spuštění by byl uživatel stále *Honza*, přesto že druhé spuštění naplánoval *Karel*.
Důvodem bylo, že v této situaci se nevolala procedura SETINVBATCH_TOBERUN. Bylo tedy nutné udělat i úpravu kódu v proceduře DoRunBatch, která se volá při spuštění batche. K části kódu, kde se aktualizuje příští čas spuštění, jsem tedy doplnil aktualizaci uživatelského jména podobně jako v předchozím případě.

5.3 Oprava chyb při importování PTI

Vzhledem k tomu, že funkcionalita importování PTI faktur je rozsáhlá a nová, bylo potřeba v ní řešit nejednu chybu. Nebyly to ani tak chyby, které by způsobovaly nějaké zásadní problémy, ale jejich oprava řešila zlepšení robustnosti aplikace. Řešil jsem dvě z těchto chyb, které jsem sice nezpracoval najednou, ale shrnu je do této jedné kapitoly.

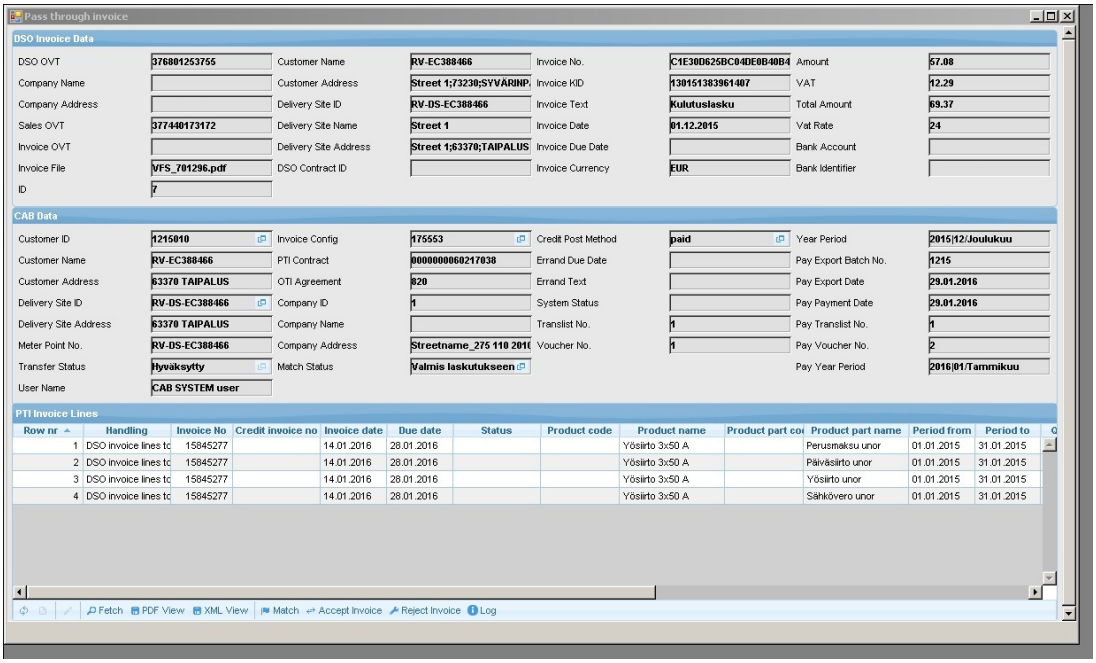
5.3.1 Popis problému

Při importování PTI faktur je potřeba validovat data v přijatých *XML* souborech, aby nedošlo k různým chybám, které by se mohly projevit na výsledné faktuře. První chybou bylo, že jed-

notlivé položky na faktuře mohly mít špatně nastaven rozsah dat tak, že mohl být nastaveno konečné datum před tím počátečním.
Druhá chyba v kontrole při importu způsobovala, že bylo možné importovat faktury, do kontraktu, který již byl vyfakturován a ukončen. Výsledkem těchto dvou chyb bylo, že se faktura naimportovala do CABu správně, i když by neměla projít validací.

5.3.2 Popis řešení

Po konzultaci s kolegy jsem se dozvěděl, že validaci dat při importu PTI faktur zajišťuje procedura pr_internal_matching. Tato procedura je poměrně dlouhá, ale dá se rozdělit na jednotlivé logické části, podle toho co ověřují. V první části se skládá ze dvou hlavních větví, které rozlišují, zda byla volána automatickým procesem nebo manuálně z grafického rozhraní. Vzhledem k tomu, že potřebuji ověřovat data v obou případech, je nutné najít vhodné místo pro ověření až za první částí. V druhé části probíhá kontrola na nulovost jednotlivých parametrů nutných ke správnému importování. V poslední a zároveň nejdelší části se kontrolují samotné položky faktury.
První chybu jsem se rozhodl řešit v poslední části, kde se kontrolují řádky faktury. Tuto kontrolu zajišťuje funkce *fn_amount_check*. Tato funkce se skládá z asi desítky dalších funkcí, které kontrolují každou položku faktury z určitého hlediska. Jednotlivé podfunkce vrací hodnotu pravda/nepravda podle toho zda položka prošla kontrolou. Celá funkce *fn_amount_check* poté vrací pouze stav kontroly. Rozhodl jsem se přidat další podfunkci, která bude kontrolovat jednotlivé položky na správnost rozsahu dat. Hlavní částí této funkce je kurzor, který kontroluje, jestli je koncové datum větší nebo rovno počátečnímu datu. Pokud ano, tak má položka nastaven chybový status a zapíše se chyba do logu. Chybový status jsem vytvořil nový a ten se promítne jako detailní chybová hláška v GUI, aby informovala uživatele.
Druhá chyba nesouvisela ani s nulovostí parametrů, ani s položkami faktury. Přesto jsem ji musel umístit do druhé části, kde se kontroluje právě nulovost atributů. Důvodem bylo udržení jednotného vzhledu kódu. Druhá část je totiž několik podmínek, které kontrolují parametry a následně nastavují *pti_match_status_id* pro danou PTI fakturu. Match status popisuje, jestli se validace dat ve faktuře povedla a pokud ne, tak podle identifikátoru a chybové zprávy lze poznat, kde nastal problém.
Řešení této chyby mi velmi ulehčilo to, že každý kontrakt má příznak invoicedone. Ten je nastaven na 1, pokud už byla vystavena poslední faktura kontraktu, takže pro daný kontrakt neexistují žádné nevyřešené závazky. Vytvořil jsem proto podfunkci v proceduře *pr_internal_matching*, která pouze kontroluje nastavení příznaku invoicedone pro daný kontrakt. Následně jsem ji volal, až poté co se kontroluje nulovost parametru *invconf_id*, aby nedošlo k vyjímce. Pokud tato funkce zjistí chybu, tak dojde k nastavení *pti_invoice.pti_match_status_id* a přeruší se další kontrolování dat. Uživatel je poté informován o chybě v grafickém rozhraní.



Obrázek 3: Testování v grafickém rozhraní aplikace PTI

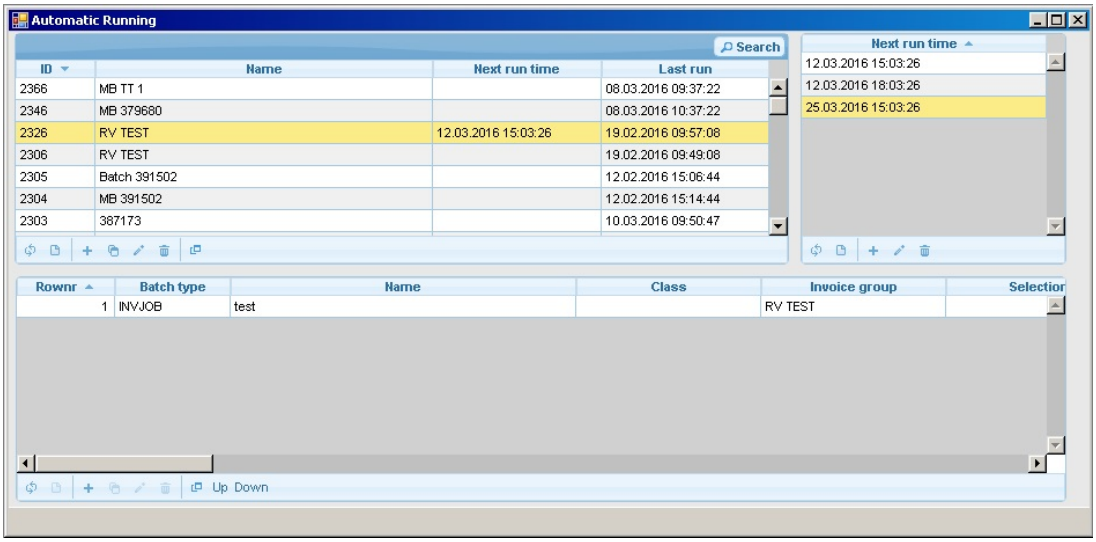
Testování obou chyb jsem prováděl pomocí transakce, ve které jsem vkládal testovací data a volal proceduru *pr_internal_matching*. V průběhu transakce jsem si vypisoval stavy proměnných, což mi nahradilo krokování. Tuto transakci jsem spouštěl na jednom z našich integračních prostředí a následným rollbackem jsem zajistil, že neovlivním žádná cizí data. Finální testování jsem prováděl pomocí grafického rozhraní, viz obr. 3, abych si ověřil správnost dat i při klasickém použití.

5.4 Změny v GUI

Mimo předchozí řešené úkoly jsem rovněž opravoval několik chyb v grafickém rozhraní. Většinou se jednalo o chyby s nízkou prioritou, které nevyžadovaly okamžitou opravu. Vzhledem k tomu, že tyto chyby byly často velmi podobné, tak uvedu pouze pár z nich a shrnu je v této jedné kapitole.

5.4.1 Oprava zobrazování součtu cen ve statistikách

Statistiky jsou součástí aplikace Invoice job a Invoice batch, které jsem již popsal výše. V těchto statistikách jsou jednoduše zobrazeny součty hodnot faktur, které se podařilo (resp. nepodařilo) spočítat v rámci určité dávky. Data jsou jednoduše načítána z databáze a podle stavu faktur jsou třízena do jednotlivých řádků.
Tato komponenta je psána ve *Visual Basicu 6* s použitím knihoven třetích stran, konkrétně *Spread* COM komponent ve verzi 6.0 od firmy *Grapecity Inc.* Chyba byla v tabulce, která vypisovala jednotlivé součty s přesností na jedno desetinné místo. Tento problém se vyskytoval



Obrázek 4: Aplikace pro správu fakturačních dávek

pouze na zákaznickém prostředí.

První jsem proto ověřil, zda je na Windows serveru u zákazníka správně nastaveno zobrazování čísel v regionálním nastavení systému Windows. Jelikož nastavení bylo správné, tak jsem přešel k opravě chyby v kódu. Dokumentaci ke knihovnám Spreadu jsem bohužel nenašel, proto jsem řešení hledal na internetových diskuzních fórech. Na oficiálním fóru *Spread*, viz [6], byla zmíněna jedna vlastnost pro sloupce v tabulce s názvem TypeFloatDecimalPlaces. Přidal jsem tedy tuto vlastnost k mé tabulce a vyzkoušel knihovnu v lokálním CABu. Testování proběhlo úspěšně, i když jsem nastavil v systému zobrazování čísel na více desetinných míst.

5.4.2 Vylepšení aplikace automatic running

Jedním z dalších úkolů bylo opravení několika chyb v aplikaci Automatic running. Tato aplikace je psaná ve firemním *CAB Frameworku*. Vzhled aplikace můžete vidět na obr. 4

V této aplikaci bylo několik chyb, které snižovaly robustnost aplikace. Jedna z těchto chyb nastala, když uživatel chtěl kopírovat invoice batch v tabulce, ale neměl označen řádek v tabulce. Při této situaci se aplikace ukončila. Řešením bylo přidání události na dané tlačítko, která zavolala funkci v *JavaScriptu*. Tato funkce jednoduše kontroluje označený řádek pomocí *jQuery* a vestavěných funkcí *CAB Frameworku* a následně otevře dialogové okno s chybovou zprávou.

Druhou chybou bylo pouze přidání lokalizací pro některé chybějící texty. Tyto texty byly napsány anglicky rovnou v kódu, což bylo potřeba změnit za zástupné klíče. Lokalizace jsem tedy řešil přidáním záznamů do databáze a následným mapováním v kódu. Po přidání daných textů jsou jednotlivé texty z databáze načteny a následně se z nich dá vygenerovat knihovna, která se pouze přidá k instalačnímu balíčku.

6 Závěr

Absolvování bakalářské praxe mi přineslo přesně to, co jsem očekával. Vyzkoušel jsem si skutečnou práci na realném projektu ve firmě. Kromě toho jsem si vyzkoušel vývoj projektu, který v sobě zahrnuje řadu různých technologií. S většinou programovacích jazyků a technologií jsem se již seznámil na škole, což bylo pro mě velkým přínosem a ulehčením v začátcích. Následně jsem si tyto vědomosti zdokonalil svou prací ve firmě. Použitím firemního *CAB Frameworku* jsem si rovněž rozšířil obzory a viděl, jak propracovaná může být technologie i když je zpracována pouze několika lidmi.

Velkým přínosem byla rovněž práce v týmu. Spolupráce byla o to zajímavější, když polovina týmu byla na druhém konci světa. Proto jsem vedle technických znalostí využil i znalosti angličtiny, kterou jsem se učil během studia.

Literatura

[1] Informace o Tieto [online]. [cit. 2015-12-18].
Dostupné z: http://www.tieto.cz/tieto-o-nas

[2] Overview of the .NET Framework [online]. [cit. 2016-04-18].
Dostupné z: https://msdn.microsoft.com/en-us/library/zw4w595w

[3] Supported Application Development Languages [online] [cit. 2016-04-17].
Dostupné z: https://docs.oracle.com/cd/B28359_01/server.111/b28318/data_access.htm#i8536

[4] What is XML (Extensible Markup Language)? - Definition from WhatIs.com [online] [cit. 2016-04-17].
Dostupné z: http://searchsoa.techtarget.com/definition/XML

[5] Introduction to Visual Basic [online] [cit. 2016-04-18].
Dostupné z: http://www.vbtutor.net/lesson1.html

[6] Topic: Rounding behavior upon setting Value property on Float cell - Spread Help [online]. [cit. 2016-04-10].
Dostupné z: http://sphelp.grapecity.com/forums/topic/rounding-behavior-upon-setting-value-property-on-float-cell/